```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from scipy import stats

import os
import pathlib
import glob

# plotting a distribution

x = np.random.rand(100000)

# x

#os.listdir()

import glob
import re

# Load unsorted filenames
filenames = glob.glob("/home/puja/Downloads/18_4/flat4/*.FIT")

# Custom sort: extract the **last number** in filename for sorting
def extract_number(f):
    match = re.search(r'(\d+)(?=\.FIT$)', f)
    return int(match.group(1)) if match else -1

# Sort based on that number
filenum = sorted(filenames, key=extract_number)

filenum
```

```
['/home/puja/Downloads/18_4/flat4/TWO001.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO002.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO003.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO004.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO005.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO006.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO007.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO008.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO009.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO010.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO011.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO012.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO013.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO014.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO015.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO016.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO017.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO018.FIT',
```

```
 '/home/puja/Downloads/18_4/flat4/TWO019.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO020.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO021.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO022.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO023.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO024.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO025.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO026.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO027.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO028.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO029.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO030.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO031.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO032.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO033.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO034.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO035.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO036.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO037.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO038.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO039.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO040.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO041.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO042.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO043.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO044.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO045.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO046.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO047.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO048.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO049.FIT',
 '/home/puja/Downloads/18_4/flat4/TWO050.FIT']
```

```
len(filenum)
```

```
50
```

```
Gain = 2.58             # e- per ADU
sigma_threshold = 4.0   # Adjust threshold for hot pixels . Should not
be too aggresive otherwise exposure correction would be affected!
binsize = 1

# [fits.getdata(f) - 100 for f in filenum]


# Read all files, stack, and apply hot-pixel mask
stacked_img = np.array([fits.getdata(f) - 100 for f in filenum])    #
read all files and correct for the PEDESTEL (check header)
stacked_img = stacked_img * Gain                                    #
Convert AUDs to electrons
```

```
stacked_img.shape
```

```
(50, 510, 765)
```

```
# Identify hot pixels in each frame
median_frame = np.median(stacked_img, axis=(1, 2), keepdims=True)    #
Per-frame mean
std_frame = np.std(stacked_img, axis=(1, 2), keepdims=True)      #
Per-frame std dev
```

```
std_frame
```

```
array([[[190.62230254]],

       [[190.5987323 ]],

       [[189.65695164]],

       [[191.36040351]],

       [[190.23110207]],

       [[192.01164227]],

       [[190.02118935]],

       [[191.317553  ]],

       [[189.62228405]],

       [[191.86054117]],

       [[190.70963345]],

       [[191.35851708]],

       [[190.98406463]],

       [[190.45285562]],

       [[189.67710824]],

       [[191.02150605]],

       [[189.64973159]],

       [[190.2408215 ]],

       [[191.03835055]],
```

```
       [[190.85840261]],
       [[189.26845499]],
       [[189.89161545]],
       [[189.68279086]],
       [[191.8433478 ]],
       [[189.61359476]],
       [[190.73316923]],
       [[189.84232916]],
       [[191.54439174]],
       [[189.69381014]],
       [[191.62559653]],
       [[189.47185579]],
       [[190.77349045]],
       [[191.45048763]],
       [[191.39529826]],
       [[189.83984015]],
       [[191.39428192]],
       [[191.05340925]],
       [[190.99898153]],
       [[191.26998183]],
       [[191.27966138]],
       [[190.68006898]],
       [[191.87016266]],
       [[191.37264022]],
       [[192.01965395]],
       [[191.79984025]],
```

```
       [[192.02450248]],

       [[191.85078628]],

       [[190.8817114 ]],

       [[191.25701972]],

       [[192.48334626]]])
```

```
# masking

hot_pixel_mask = stacked_img > (median_frame + sigma_threshold *
std_frame)  # Find hot pixels
stacked_img_filtered = np.where(hot_pixel_mask, np.nan, stacked_img)
# Replace hot pixels with NaN



#-------------------------------------------------
# np.where(condition, x, y) works as follows:
#If condition is True, it selects values from x.
#If condition is False, it selects values from y.
#-------------------------------------------------

# np.savetxt('boolean.txt', hot_pixel_mask[0])

xx = np.array([11, 14, 66, 171, 12, 6143])
np.where(xx == 171)
```

```
(array([3]),)
```

```
ntot = stacked_img_filtered.size  # Total number of pixels
nhot = np.isnan(stacked_img_filtered).sum()  # Count NaNs

nhot
```

```
11108
```

```
print("Number of masked pixels: %i (%5.3f%%)" % (nhot, (nhot * 100 /
ntot)))
#print(np.isnan(stacked_img_filtered).sum())
```

```
Number of masked pixels: 11108 (0.057%)
```

```
reference_frame = np.nanmedian(stacked_img_filtered, axis=0)  # Median
frame served as reference

reference_frame
```

```
array([[23349.  , 23301.27, 23229.03, ..., 23329.65, 23468.97,
23416.08],
```

```
       [23527.02, 23439.3 , 23410.92, ..., 23532.18, 23532.18,
23599.26],
       [23523.15, 23474.13, 23279.34, ..., 23356.74, 23441.88,
23587.65],
       ...,
       [23435.43, 23416.08, 23276.76, ..., 23309.01, 23478.  ,
23532.18],
       [23537.34, 23439.3 , 23501.22, ..., 23484.45, 23475.42,
23609.58],
       [23674.08, 23645.7 , 23576.04, ..., 23583.78, 23648.28,
23621.19]])
```

```
scale_factors = np.nansum(reference_frame) /
np.nansum(stacked_img_filtered, axis=(1, 2), keepdims=True)
stacked_img_filtered_expcorr = stacked_img_filtered * scale_factors
# exposure corrected images

# scale_factors
stacked_img_filtered_expcorr
```

```
array([[[23400.83486518, 23416.46494524, 23247.13907791, ...,
         23437.30505199, 23460.75017208, 23481.59027883],
        [23327.89449156, 23546.71561242, 23533.6905457 , ...,
         23502.43038558, 23447.72510536, 23729.06654647],
        [           nan, 23705.62142638, 22984.03273021, ...,
         23249.74409125, 23512.85043895, 23614.44595935],
        ...,
        [23346.12958496, 23505.03539892, 23510.24542561, ...,
         23288.81929141, 23314.86942484, 23471.17022545],
        [23163.77865092, 23489.40531886, 23200.24883772, ...,
         23726.46153312, 23327.89449156, 23624.86601272],
        [23588.39582592, 23656.12617285, 23346.12958496, ...,
         23739.48659984,            nan, 23726.46153312]],

       [[23324.26993721, 23446.29138501, 23220.42189653, ...,
         23435.90658094, 23542.35082264, 23550.13942569],
        [23690.33428062, 23433.31037993, 23438.50278196, ...,
         23581.2938379 , 23622.83305417,            nan],
        [23586.48623993, 23342.44334433, 23082.82324262, ...,
         23617.64065214, 23674.75707451, 23731.87349689],
        ...,
        [23493.02300332, 23557.92802875, 23223.01809755, ...,
         23345.03954535, 23412.54077179, 23495.61920434],
        [23563.12043078, 23464.46479213, 23389.17496264, ...,
         23446.29138501, 23464.46479213, 23547.54322468],
        [23692.93048163,            nan, 23711.10388875, ...,
                    nan, 23568.31283281,            nan]],

       [[23085.13359659, 23124.00182018, 23212.10312698, ...,
         23121.41060527, 23486.77190701, 23463.45097285],
```

```
       [23427.17396417, 23372.75845115, 23510.09284116, ...,
        23626.69751192, 23572.2819989 , 23546.36984984],
       [23567.09956909, 23538.59620512, 23481.58947719, ...,
        23188.78219283, 23453.08611323, 23732.93732306],
       ...,
       [23453.08611323, 23414.21788964, 23118.81939037, ...,
        23326.11658284, 23331.29901265, 23595.60293305],
       [           nan, 23460.85975795,            nan, ...,
        23377.94088096, 23502.31919644, 23730.34610816],
       [           nan, 23740.71096778, 23707.02517401, ...,
                   nan,            nan, 23655.20087589]],

      ...,

      [[23571.07897832, 23396.09716931, 23679.155978  , ...,
        22966.36243248, 23483.58807381, 23596.81159729],
       [23293.16669342, 23450.13566915,            nan, ...,
                   nan, 23336.91214567,            nan],
       [23455.28219294, 23426.97631207, 23457.85545484, ...,
        23524.76026417, 23710.03512077, 23545.34635934],
       ...,
       [23321.47257429, 23357.49824085, 23372.93781223, ...,
        23288.02016963, 23421.82978828, 23537.62657365],
       [23594.23833539, 23630.26400195, 23275.15386014, ...,
        23689.44902559, 23259.71428876, 23648.27683523],
       [23699.74207318,            nan, 23604.53138298, ...,
        23576.22550211,            nan,            nan]],

      [[23239.93785089, 23363.54084445, 23378.99121865, ...,
        23453.66802725, 23471.69346381, 23453.66802725],
       [23584.99620791, 23242.51291326, 23561.82064661, ...,
        23384.14134338, 23546.37027242, 23572.12089608],
       [23399.59171757, 23458.81815198, 23288.86403584, ...,
        23167.83610465, 23711.17426383, 23703.44907673],
       ...,
       [23479.41865091, 23734.34982512, 23255.38822509, ...,
        23149.81066809, 23252.81316272, 23680.27351544],
       [23713.74932619, 23456.24308962, 23440.79271542, ...,
        23536.07002296, 23402.16677994,            nan],
       [23492.29396274, 23639.07251759, 23610.74683156, ...,
        23541.22014769,            nan,            nan]],

      [[23167.76674715, 23450.8654299 , 23144.60412765, ...,
        23113.72063498, 23427.7028104 , 23700.50699561],
       [23656.755381  ,            nan, 23474.0280494 , ...,
        23476.60167379, 23600.13564445, 23548.66315667],
       [23597.56202006, 23633.5927615 , 23548.66315667, ...,
        23136.88325448, 23497.1906689 , 23437.99730796],
       ...,
       [23561.53127862, 23479.17529818, 23497.1906689 , ...,
```

```
        23378.80394702, 23373.65669824, 23679.9180005 ],
       [23661.90262978, 23399.39294213, 23649.03450783, ...,
        23677.34437611, 23569.25215178, 23628.44551272],
       [           nan, 23654.18175661, 23468.88080062, ...,
                   nan,            nan, 23584.69389812]]])

# Correct for non-uniform illumnination / fixed pattern -- flat
fielding

master_flat = np.nanmedian(stacked_img_filtered_expcorr, axis=0)    #
Compute the median frame
master_flat[np.isnan(master_flat)] = 1                              #
Avoid division by NaN
norm_master_flat = master_flat/np.nanmedian(master_flat)            #
Normalized the median frame by median

norm_master_flat

array([[1.01523496, 1.01563571, 1.01030184, ..., 1.01549042,
1.02077125,
        1.0199085 ],
       [1.02256073, 1.01998821, 1.01865184, ..., 1.02318032, 1.0235553
,
        1.02736339],
       [1.02396252, 1.02027291, 1.01139788, ..., 1.01474169,
1.02097221,
        1.02661073],
       ...,
       [1.01924659, 1.01690574, 1.0125363 , ..., 1.01425601,
1.01960826,
        1.02304031],
       [1.02415831, 1.01951352, 1.02103416, ..., 1.02110967,
1.02011861,
        1.02684263],
       [1.02980102, 1.02979673, 1.02589337, ..., 1.0256035 , 1.0290299
,
        1.02737378]])

stacked_img_filtered_expcorr_norm =
stacked_img_filtered_expcorr/norm_master_flat   # normalize the frames
by master flat

stacked_img_filtered_expcorr_norm.shape

(50, 510, 765)

frameid = np.random.randint(1,len(filenum))
frameid

47
```

```
print(frameid, filenum[frameid])
ele = stacked_img_filtered_expcorr_norm[frameid,:,:]
electron = ele.flatten()
```

```
47 /home/puja/Downloads/18_4/flat4/TWO048.FIT
```

```
electron
```

```
array([23217.36329647, 23035.91422007, 23437.70448955, ...,
       22987.66082758,            nan,            nan])
```

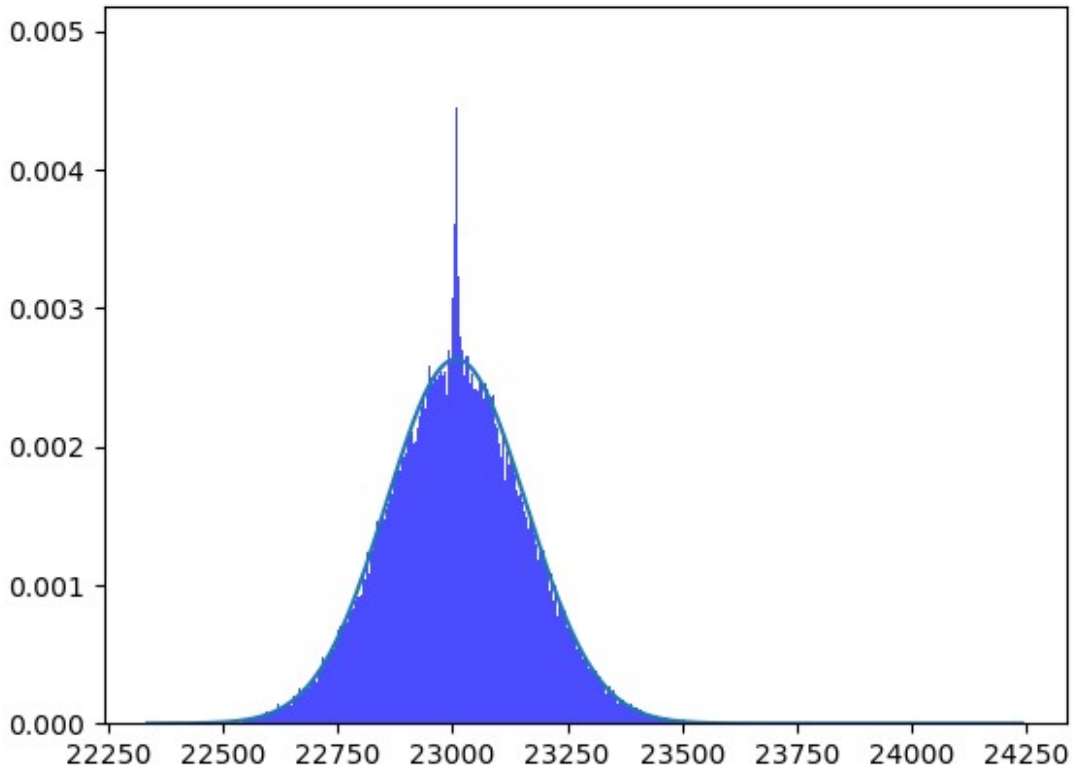
```
mine = min(electron)
maxe = max(electron)
meane = np.round(np.nanmean(electron))
vare = np.round(np.nanvar(electron))
sige = np.round(np.nanstd(electron))
```

```
mine, maxe, meane, vare, sige
```

```
(22335.188554612894, 24240.52992723784, 23008.0, 22835.0, 151.0)
```

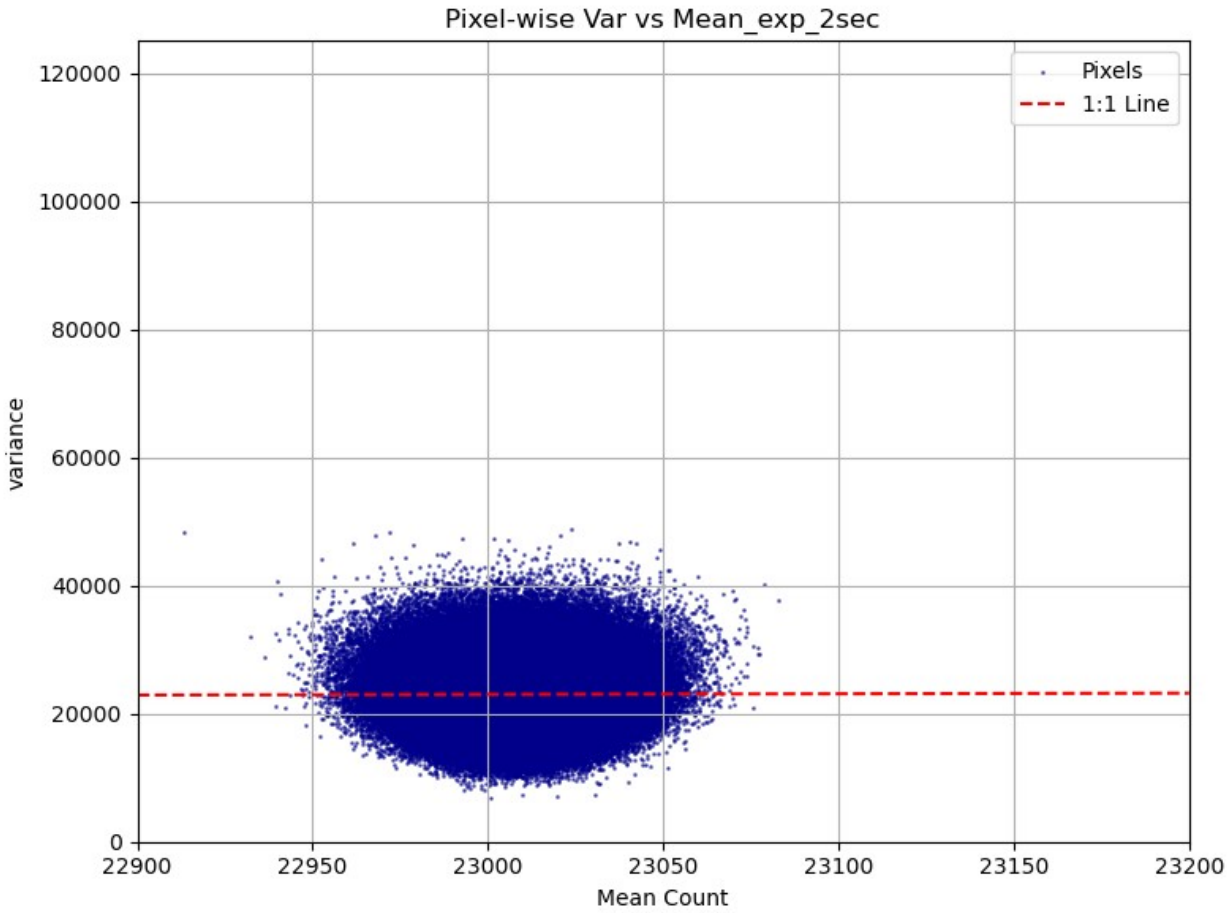
```
bins=np.round(np.arange(mine,maxe+binsize,binsize))
print(len(bins))
yy = plt.hist(electron,bins=bins,alpha=0.7,density=True, color="blue")

#bin_centers = (yy[1][:-1] + yy[1][1:]) / 2  # Midpoints of histogram
bins
yy_mod = stats.poisson.pmf(bins,mu=meane)
plt.plot(bins, yy_mod)
plt.show()
```

```
1907
```




```
def gauss(x, mu, sigma):
    return 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-(x-
mu)**2/(2*sigma**2))

mine = min(electron)
maxe = max(electron)
meane = np.round(np.nanmean(electron))
vare = np.round(np.nanvar(electron))
std = np.round(np.nanstd(electron))

a=np.nanmedian(electron)
b=np.linspace(0,0.005,10)
c=len(b)
d=np.ones(len(b))*a
plt.scatter(d,b)
yy = plt.hist(electron,bins=bins,alpha=0.7,density=True, color="blue")

plt.plot(bins,yy_mod,label="Model Poisson PDF",color="red")
plt.xlabel("Number of e-")
plt.ylabel("PDF")
plt.title(f"spatial distribution_exp_2sec, Mean = {meane}, Std =
{sige}, Var = {vare}")
outfilenum = "pixel_flux_dist_"+str(frameid)+".png"
plt.savefig("spatial_exp_4",dpi=200)
plt.legend()
```

```
print(f"Figure saved as: {outfilenum}")
#plt.show()

import matplotlib.pyplot as plt
import numpy as np
data = stacked_img_filtered_expcorr_norm
# Compute per-pixel mean and sigma (over time axis)
mean_map = np.mean(data, axis=0)   # Shape: (height, width)
sigma_map = np.std(data, axis=0)   # Shape: (height, width)

# Assuming `mean_map` and `sigma_map` are already defined
mean_flat = mean_map.flatten()
sigma_flat = sigma_map.flatten()
var=(sigma_flat)**2
#x_0=np.nanmedian(var)
#y_0=np.nanmedian(mean_flat)
plt.figure(figsize=(8, 6))
plt.scatter(mean_flat, var ,s=1, alpha=0.5,
label='Pixels' ,color='darkblue')



plt.xlabel('Mean Count')
plt.ylabel('variance')
plt.title('Pixel-wise Var vs Mean_exp_2sec')

plt.ylim(0, 125000)
plt.xlim(22900, 23200)

# Add full-range 1:1 line
x_min, x_max = 22900, 23200
plt.plot([x_min, x_max], [x_min, x_max], 'r--', label='1:1 Line')

plt.legend()
plt.grid(True)
plt.tight_layout()
# Save the plot
plt.savefig("temporal_exp4.png", dpi=300)

plt.show()
```


Pixel-wise Var vs Mean_exp_2sec
Pixels
1:1 Line
variance
Mean Count
0
20000
40000
60000
80000
100000
120000
22900
22950
23000
23050
23100
23150
23200